MiNT-m: An Autonomous Mobile Wireless Experimentation Platform

Pradipta De*, Ashish Raniwala, Rupa Krishnan, Krishna Tatavarthi, Jatan Modi, Nadeem Ahmed Syed, Srikant Sharma, and Tzi-cker Chiueh
Department of Computer Science
Stony Brook University, Stony Brook, New York 11794
{prade, raniwala, krishnan, krishnak, jatan, nadeem, srikant, chiueh}@cs.sunysb.edu

ABSTRACT

Limited fidelity of software-based wireless network simulations has prompted many researchers to build testbeds for developing and evaluating their wireless protocols and mobile applications. Since most testbeds are tailored to the needs of specific research projects, they cannot be easily reused for other research projects that may have different requirements on physical topology, radio channel characteristics or mobility pattern. In this paper, we describe the design, implementation and evaluation of MiNT-m, an experimentation platform devised specifically to support arbitrary experiments for mobile multi-hop wireless network protocols. In addition to inheriting the miniaturization feature from its predecessor MiNT [9], MiNT-m enables flexible testbed reconfiguration on an experiment-by-experiment basis by putting each testbed node on a centrally controlled untethered mobile robot. To support mobility and reconfiguration of testbed nodes, MiNT-m includes a scalable mobile robot navigation control subsystem, which in turn consists of a vision-based robot positioning module and a collision avoidance-based trajectory planning module. Further, MiNT-m provides a comprehensive network/experiment management subsystem that affords a user full interactive control over the testbed as well as real-time visualization of the testbed activities. Finally, because MiNT-m is designed to be a shared research infrastructure that supports 24x7 operation, it incorporates a novel automatic battery recharging capability that enables testbed robots to operate without human intervention for weeks.

Categories and Subject Descriptors: C.3 Special-Purpose and Application-based Systems: Miscellaneous; C.2.3 Network Operations: Network Management, Network Monitoring [Wireless Testbed]
General Terms: Experimentation, Management, Measurement, Design.
Keywords: Wireless Experimentation Platform, Topology Reconfiguration, Mobility, Autonomous Operation.

*Currently affiliated as a Research Staff Member with IBM India Research Laboratory, Hauz Khas, New Delhi 110 016 (pradipta.de@in.ibm.com)

MobiSys'06, June 19–22, 2006, Uppsala, Sweden.

1. INTRODUCTION

Software-based wireless network simulations often fail to faithfully capture many real-world radio signal propagation effects such as non-uniform path loss, interference, and multipath fading [11]. This limitation is overcome through use of physical wireless network testbeds. Although several ongoing efforts aim to construct a high-fidelity wireless network testbeds for wireless protocol experimentation, they share some of the following weaknesses. First, these testbeds provide limited flexibility to the user to specify arbitrary initial wireless connectivity with user-defined pairwise signal-to-noise ratios (SNR), as well as, node mobility patterns during experiments. Second, often support for extensive node mobility is accompanied with significant manual maintenance efforts for node charging and positioning, and thus cannot work as an autonomic infrastructure that supports 24x7 operation. Finally, management and control interfaces of many existing wireless network testbeds are often rather primitive. Experiences of several groups in setting up wireless experiments underscore the need for a comprehensive monitoring/control environment to improve the productivity of wireless protocol experimentation, especially for testing cases involving node mobility and failures. This paper describes the design, implementation, and evaluation of *MiNT-mobile (MiNT-m)*, a multi-hop mobile wireless network testbed that (1) supports experiment-by-experiment topology reconfigurability, (2) enables untethered node mobility, (3) can operate autonomously in a 24x7 fashion, (4) provides comprehensive monitoring and control of the network testbed, (5) supports hybrid ns-2 simulations[1], and (6) is deployable in a limited physical space using radio signal attenuation.

An obvious way to support node mobility and topology reconfiguration in a wireless network testbed is to mount each testbed node on a mobile robot. Though conceptually simple, there are several technical challenges in designing and implementing such a wireless testbed. First, each testbed node must be battery-operated and self-rechargeable. The key design issue here is how to build completely untethered mobile robots that can operate *autonomously*, thereby far

[1]Unless otherwise specified, we use the term simulation and hybrid simulation interchangeably.

exceeding in usability the ones that are simply battery operated and require frequent management. Second, to set up a given initial topology or to enact a particular run-time node movement pattern, an accurate positioning mechanism is required to track and control the position of each wireless network testbed node. Finally, to grow a mobile wireless network testbed to a significant size of about 100 nodes, the targeted size of the MiNT-m project, the cost of each testbed node must be low, and the design of various testbed control functions, such as node movement and position tracking, must be scalable.

A MiNT-m node is built using a low-cost commodity robotic vacuum cleaner called Roomba [1], which supports a limited number of externally controllable movements (move forward, and change direction), is able to carry a large payload (up to 30 pounds), and comes with an effective autorecharging capability. Mounted on each Roomba is a wireless network node supporting four 802.11 interfaces, each of which is attached to an antenna through a radio signal attenuator to reduce its signal coverage. Roomba's autorecharging circuitry is modified to power up both Roomba and the wireless network node. Moreover, a *residual power estimation* and a *recharge scheduling algorithm* is designed to keep track of the battery status of each node and determine the next recharge time for a node. A *vision-based positioning system* is designed to track the position of each mobile node in the testbed. The positioning system robustly tracks the nodes with zero false positives, and requires only commercial off-the-shelf webcams. The resulting node position estimates are used for monitoring and for *planning trajectories for collision-free node movement.*

A network/experiment management system is essential to the robustness and usability of any wireless network testbed. Existing wireless network simulators or testbeds neither provide real-time visibility into the detailed dynamics of the protocols under simulation, nor support sufficient control flexibility for steering the simulation runs towards more fruitful directions. The network/experiment management system designed for MiNT, called MOVIE (*Mint-m cOntrol and Visualization InterfacE*) bridges this deficiency. Specifically, MOVIE provides *real-time* display of network traffic load distribution, pair-wise end-to-end routes, node/link liveliness, protocol-specific state variables, positions of individual nodes and inter-node signal-to-noise ratios. In addition, MOVIE allows users to control a simulation run dynamically, including *pausing* a simulation run at a user-specified breakpoint, inspecting its internal states and/or network conditions, modifying different simulation parameters, and resuming the run. It also supports a *rollback* mechanism that allows one to go back to a previous state of a long-running simulation, and resume from there with a different set of simulation parameters.

In summary, this paper makes the following key contributions to the state of the art of wireless network protocol simulation systems:

- MiNT-m provides the most flexible testbed topology reconfigurability and node mobility support for protocol simulations among all existing wireless network testbeds. In addition, MiNT-m's node mobility infrastructure requires no manual configuration in (a) node movement control, (b) node position tracking and (c) node recharging. This reduces the testbed setup and administration cost to a minimum.
- MiNT-m offers one of the most advanced management interfaces among all existing wireless protocol simulation systems. This interface, called MOVIE, is able to provide a detailed real-time view of the system configuration, network traffic load distribution, node/link liveliness, and evolution of protocol-specific states. In addition, MOVIE provides users the flexibility to dynamically steer the direction of a simulation run (including reversing the execution) by inspecting protocol states and modifying protocol parameters, network configurations, and traffic loads accordingly.

The rest of the paper is organized as follows. Section 2 discusses prior work related to this research. Section 3 gives an overview of MiNT-m. Section 4 discusses the details of the mobility infrastructure. Section 5 presents the techniques to enable 24x7 autonomous operation of the testbed, while Section 6 details the MOVIE implementation. Section 7 evaluates various aspects of the first 12-node MiNT-m prototype, and records results of experiments performed using the prototype. Finally, Section 8 concludes the paper with a summary of contributions.

2. RELATED WORK

In this section, we compare the mobility and visualization aspects of MiNT-m with those of other wireless network testbeds/simulators.

2.1 Node Mobility

Volunteer supported node mobility have been tried in an early wireless testbed at CMU [15], as well as by the APE testbed at Uppsala University [14]. Ideally, node movements should be remotely controlled and their mobility fully automated, instead of depending on volunteers. In this paper, we take the approach of introducing mobility using remotely controlled robots.

The ORBIT testbed [18] introduces *virtual mobility* using *fixed* wireless nodes by transferring states of a virtual node from one physical node to another. This leads to discretized mobility.

Mobile Emulab [12] uses 4 Acroname Garcia robots for mobility. These robotic platforms cost over $1000 a piece, as opposed to our improvised robotic platform based on Roomba robotic vacuum cleaners ($249 a piece) [1]. Moreover, Netbed's robots must be manually taken to their charging bases every 2-3 hours for recharge. Roomba comes with an auto-charging feature, that makes our mobile testbed truly autonomous. We also leverage on an earlier design of using attenuators to miniaturize the testbed that we proposed in MiNT [9]. This makes the arena of operation smaller thus requiring smaller number of overhead cameras to track the nodes.

A key aspect in node mobility is to allow collision-free movement of the robots, which requires path and motion planning of the robots. Existing literature [6] has explored robot motion planning for various complex scenarios. Since our testbed offers a much controlled environment, we explore a heuristic (discussed later) that is lightweight, and computationally efficient. In contrast to the motion planning algorithm used in Mobile Emulab, since the Roombas do not have object sensor, we have built the intelligence to detect obstacles present on the path into the trajectory planning procedure by using the tracking subsystem.

2.2 Node Tracking

Associated with mobility feature is the use of a tracking system for accurately determining the position and orientation of each node. There are various systems that use vision-based tracking system to track mobile nodes in different environments. Here, we briefly discuss three tracking systems that are most similar to ours. Graham and Kumar [10] use ceiling-mounted cameras and colored patterns on toy cars to track them. They use 8 colors and an error-correcting 3x2 colored pattern to track the cars. Their system is designed to track up to 22 mobile nodes and is able to uniquely identify nodes as well as provide their position and orientation. Cremean et al. [7] use ceiling-mounted monochromatic camera and binary (black/white) patterns to compute the position and orientation of the nodes. Very recently and concurrent to our work, Johnson et al. [12] have also implemented a centralized tracking system that uses ceiling mounted cameras and color patterns to determine the position and orientation of the mobile nodes in their wireless testbed. Their tracking system does not uniquely identify each tracked node, instead locality and motion pattern information is used to determine the identity of nodes.

Simplicity and cost of construction are the main differentiators of our tracking system. Use of off-the-shelf consumer webcams and standard APIs makes our tracking system inexpensive and easily portable. Additionally, miniaturization makes our tracking system more scalable. We also do not need any frame-grabber cards. Thus no specialized equipment/API needs to be procured and set up to install our tracking system.

2.3 Testbed Visualization

A visualization tool is essential to study the dynamics of the experiments in a testbed. CMU testbed [15] uses a graphical interface that displays the position of the nodes, the link characteristics, route changes for DSR protocol, and throughput information. Similarly, the MIT RoofNet [5] also has an online map of the link characteristics among all the nodes in the testbed. It refreshes the link characteristics periodically. Similarly, Kurkowski et al. [13] extended NAM to develop a tool called iNSpect, which adds features needed to study the mobility of nodes. Our extension of NAM (MOVIE) goes much beyond iNSpect in making NAM display real time as well as showing important events related to wireless protocol evaluation, like route changes, link characteristics, and protocol-specific attributes. Additionally, MOVIE supports advanced control features such as experiment breakpoint based on specified events, and rollback of experiments.

3. SYSTEM OVERVIEW

MiNT-m derives from MiNT [9] the feature of using radio signal attenuation to shrink physical space. The improvement is in designing completely untethered nodes, that was lacking in MiNT due to use of desktop PCs as testbed nodes. More specifically, MiNT-m mounts a battery-powered small-form-factor embedded computing board (RouterBOARD) on an iRobot's Roomba robotic vacuum cleaner. Figure 1 shows a picture of the current 12-node MiNT-m prototype. In this section, we describe the hardware and software components of MiNT-m, and how these are integrated into a powerful and flexible wireless research platform.

Figure 1: *MiNT-m prototype with 12 mobile nodes and charging stations (top left corner of the image).*

3.1 MiNT-m Architecture

The hardware and software components in MiNT-m are shown in Figure 2, and described in more detail in this subsection.

Hardware Components: A *mobile node* comprises of a wireless computing device and a mobile robot for physical movement. In MiNT-m, the wireless device is RouterBOARD 230, which is a low-power battery-operated board. Each RouterBOARD has 4 mini-PCI IEEE 802.11 a/b/g cards, which make it possible to support multi-radio experiments [4]. A radio signal attenuator is inserted between a wireless interface and its antenna to shrink the signal coverage and thus the physical space requirement. The mobile robot is an inexpensive off-the-shelf robotic vacuum cleaner from iRobot, called Roomba. Necessary modifications to the Roomba are made to allow (i) the Roomba movements to be controlled from the wireless computing board mounted on it, and (ii) automatic recharging of a mobile node when the batteries drain out. The mobile node design is discussed in detail in Section 4.1.

The *control server* is a PC equipped with 3 wireless network interfaces. All control traffic is transported on an IEEE 802.11g channel and thus does not interfere with IEEE 802.11a channels, that are used in actual experiments. Multiple NICs allow the flexibility to scale the testbed to increasing number of testbed nodes. The *tracking server* is a cluster of PCs (currently 3 PCs) that periodically receives snapshots of the entire testbed, as captured by a (3x2) grid of commodity web cameras, and uses them for testbed node identification and positioning. Smaller physical space requirement also reduces the number of cameras needed.

Software Components: The key software components in MiNT-m are: (a) the *control daemon* running on the central control server, (b) the *node daemon* residing on each testbed node, and (c) the network monitor and control interface called MOVIE (Mint-m cOntrol and Visualization InterfacE).

The control daemon running on the control server collects position updates of testbed nodes from the tracking server and event traces from experiment nodes, and correspondingly updates the MOVIE display. It also communicates user-issued control commands, regarding node position or configuration changes, to individual node daemons that in turn control the movement of mobile robots. Because all event messages from the testbed nodes pass through the control server, the control daemon also maintains a com-

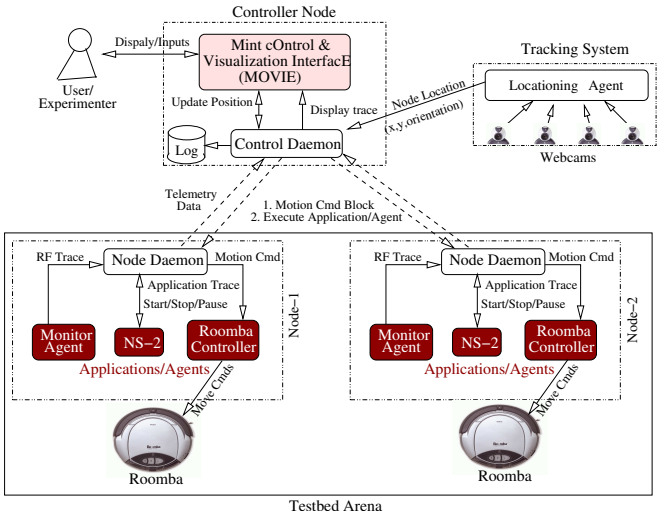

Figure 2: *The* control daemon *running on the control server collects inputs from the tracking server and the user, and controls the movement of mobile robots. It also includes the MOVIE interface for monitoring and control. Each testbed node corresponds to a Roomba robot and has a* node daemon *running on it, which communicates directly with the control daemon over a dedicated wireless control channel. The* vision-based tracking server *periodically captures images of testbed nodes, and processes them to derive the location of each testbed node.*

plete log of activities in the testbed.

The node daemons on the testbed nodes communicate with the central control daemon over an IEEE 802.11g channel that is determined at start-up time. The messages that are communicated are either movement commands from the central control daemon, or simulation events reported by testbed nodes back to the central control server. Other programs running on testbed nodes, for example, an ns-2 simulator, a TCP sender, or an RF monitoring agent, rely on the node daemon for any communications with the central control server. For example, critical events in the event trace that an ns-2 simulation run generates are passed in real time through the node daemon to the controller node for display.

MOVIE provides a comprehensive monitor and control interface that offers real-time visibility into the testbed activity and supports full interactive control over testbed configuration and hybrid simulation runs. MOVIE is derived from Network Animator (NAM), a well-known off-line visualization tool for ns-2 traces, but introduces several powerful features for real-time monitoring and controlling simulation runs and for interactively debugging simulation results such as protocol-specific breakpoints and state rollback.

3.2 Using MiNT-m

Running a hybrid simulation on MiNT-m generally involves three steps: experiment configuration, experiment execution and experiment analysis.

Experiment Configuration:
To configure an experiment running on MiNT-m, a user could specify (1) the testbed topology, (2) the applications to run on the testbed nodes, (3) mobility patterns of testbed nodes, and (4) network interface card parameters such as radio channel, transmission power, etc. MOVIE allows configuring the network topology through simple drag of a node icon in the canvas. The control daemon accordingly triggers physical movement of the chosen node, followed by update of the pairwise signal strengths in MOVIE.

When the user runs an ns-2 simulation on the MiNT-m testbed, an ns-2 instance runs on each testbed node. In order to use MiNT-m as a protocol development platform, Linux implementations of the protocol can be installed and executed on each testbed node.

To describe node mobility pattern, the user specifies the intermediate positions and final destinations, along with their relative temporal offsets with respect to the beginning of the simulation run. From these information, instead of statically computing a global trajectory for each moving testbed node, MiNT-m relies on a run-time collision avoidance algorithm that dynamically resolves possible collisions among testbed nodes by halting some of them when collisions become imminent.

The user can also configure individual testbed nodes: One can first gain a root shell on individual nodes and then deploy applications or kernel modules, and then change their wireless network card parameters, such as transmit power and retry count.

Experiment Execution:
The user initiates the experiment through MOVIE and controls the execution of an experiment by observing its progress and intermediate results. In addition to starting/stopping an experiment, MiNT-m provides users the ability to temporarily pause an experiment, modify simulation parameters on the fly, and then resume the experiment. Moreover, MiNT-m supports the ability to rollback an experiment back to a previous specified time, modify some simulation parameters and restart the simulation run from the restored state. Finally, MiNT-m borrows from VirtualWire [8] and provides a facility to introduce controlled faults that are designed to expose potential bugs in protocol implementations.

Experiment Analysis:
MiNT-m allows the user to specify simulation events of interest and to request the associated values to be displayed in real time. In addition, MOVIE supports real-time display of several wireless network parameters that are generally useful across all wireless protocols, such as the inter-node signal-to-noise ratio, the throughput on each wireless link, and route between a pair of nodes.

4. AUTONOMOUS NODE MOBILITY

Building a complete infrastructure to support mobility constitutes the most challenging part of the testbed design. The key components of the mobility infrastructure are: (a) fully mobile nodes each composed of a battery-operated wireless device mounted on a mobile robot, (b) the mechanism to keep track of the nodes' positions inside the testbed, and (c) the mechanism to enable smooth mobility of nodes. This section presents each of these components and the challenges in building them.

4.1 Mobile Testbed Node Design

Logically, each MiNT-m testbed node is a wireless networking device mounted on a mobile robot. One critical factor is cost because MiNT-m is planned to scale to a size with an order of hundred nodes. Next, for mobility, the wireless networking device should have a small form factor so that it can be easily mounted on a simple robot, and

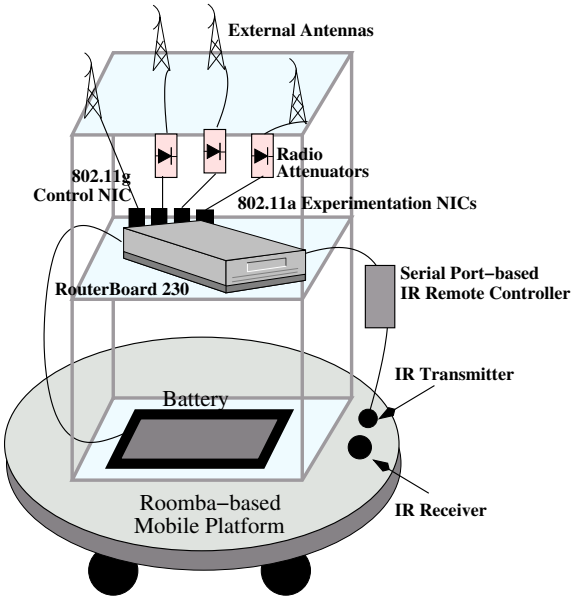

Figure 3: *A MiNT-m testbed node comprises of a RouterBOARD (RB-230) powered by an external laptop battery, and a Roomba robotic vacuum cleaner whose movement is controlled by a Spitfire Universal Remote Controller. The RouterBOARD is equipped with 4 wireless NICs each connected to a separate omni-directional antenna via a radio signal attenuator.*

should be power efficient to maximize its runtime even on a small battery. After considering several options, RouterBOARD's RB-230 board is chosen as the hardware platform for the wireless networking device. RB-230 is a small-form-factor PC with a 266MHz processor and runs on an external laptop battery. It also comes with a PCI extension board (RB-14), which allows to connect 4 Atheors-based 802.11 a/b/g mini-PCI cards. Each of these cards is connected to a 2 dBi external antenna through a 22 dB attenuator. This adds a total of 44 dB attenuation on the signal path from transmitter to receiver and thus makes it possible to deploy a 12-node MiNT-m prototype within a space of 132.75" X 168.75" (Figure 1). In addition to the fixed attenuation, the transmit power on the mini-PCI cards can be altered by 20 dBm to provide additional flexibility in tuning inter-node signal-to-noise ratio.

The mobile robot used in MiNT-m is iRobot's robotic vacuum cleaner called Roomba [1]. Unlike other commercially available robotic platforms, whose price is in the range of thousands of dollars, Roomba is much less expensive (retail price: $249) because it is a consumer-grade product. Use of Roomba greatly reduces the per-node cost and substantially improves MiNT-m's economic viability.

Designed primarily to be a vacuum cleaner, Roomba does not have an open API for controlling its movements. This limitation is circumvented through a clever use of its IR-based remote control facility. Two primitives are enough for arbitrary Roomba movement: (1) move the mobile robot forward, and (2) turn the robot by a specified angle. A Roomba can be instructed to perform these primitives through a remote controller. Roomba's remote control codes are learnt using a programmable remote controller called Spitfire [3]. The central control server moves a testbed node by sending a movement command to the testbed node's RouterBOARD, which then sends a corresponding command to Spitfire over its serial port. Eventually Spitfire issues the associated infrared code to instruct the node's Roomba to move accordingly.

Figure 3 shows the current MiNT-m testbed node prototype. There are two shelves mounted on the Roomba. The laptop battery and the Spitfire universal remote controller sit on the lower tier, while the RouterBOARD based wireless networking device is on the top tier. The four external antennas are mounted on poles located at four corners.

4.2 Position and Orientation Tracking

To enable autonomous robot movement, the central control daemon must keep track of the current position and orientation of each testbed node. One option is to use RF/ultrasound-based indoor local positioning systems such as Cricket[16]. However, this option increases the per-node cost, and introduces additional RF interference. Therefore, we choose an optical or vision-based position/orientation tracking system that only requires off-the-shelf webcams and color patches mounted on testbed nodes. The resulting tracking system is able to uniquely identify each testbed node, and accurately pinpoint its (X, Y) position and orientation (θ). Moreover, it can scale to over 100 nodes, which is the target size of MiNT-m design.

Compared with general object tracking, the object tracking problem in MiNT-m is much less complicated because of the following simplifications. First, change in lighting condition in the testbed room is infrequent. Consequently, once the color profiles have been calibrated for the initial lighting condition, it is not necessary to dynamically account for fluctuation in lighting condition. Second, colors chosen for tagging the testbed nodes are different from the background color, in this case the floor's color. The current MiNT-m prototype uses a simple color recognition algorithm that can reliably identify 8 distinct colors. Including multiple colors in each pattern used to identify a testbed node allows the scheme to scale to large number of uniquely identifiable patterns. Finally, placement of webcams that periodically take pictures of the testbed nodes does not change once it is mounted. The current MiNT-m prototype uses 6 ceiling-mounted webcams whose image planes overlap and are parallel to the floor. Each webcam continuously feeds captured images to the tracking server it is connected to over a USB port, and one tracking server supports up to two webcams.

The current MiNT-m testbed covers a floor space of 132.75" X 168.75". The webcam it uses is Logitech QuickCam 4000, whose image resolution is 320 X 240. Each webcam is placed at a height of 9.1 ft from the ground, and is able to cover a floor region of approximately 87" X 66" which means each pixel corresponds to 0.075 square inch area. To cover the entire testbed arena, the prototype uses 6 webcams. These webcams are placed such that they overlap with one another and the overlap area is large enough to hold a Roomba completely. As a result, every Roomba is completely captured by at least one webcam and the image streams from the 6 webcams can be processed independently.

MiNT-m uses colors to identify each testbed node and its position/orientation. Colors are represented using the HSV (Hue, Saturation, Value) space because the distribution of colors is more uniform, and the Hue and Saturation components are orthogonal to the Value (or brightness) compo-

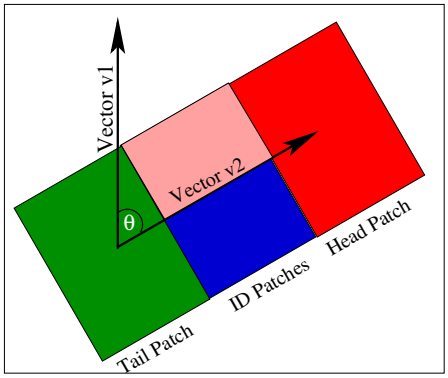

Figure 4: *The color patch on a node has a unique head and tail patch on all nodes. The vector from the centroid of the tail patch to the centroid of the head patch is used to determine the Roomba's direction, thereby computing a node's orientation in the testbed arena. The node location and identification are done using the center ID patches.*

nent. The HSV profile of a number of colors in the testbed room is computed, and eventually 8 colors are identified that are clearly distinguishable based on at least one of the H, S, or V components. Because the HSV profile for the same color may change substantially from one camera to another (due to slight changes in lighting in different parts of the room), each camera is separately profiled.

MiNT-m associates a four-color pattern with each testbed node, as shown in Figure 4. The head and tail color patches are the same for all testbed nodes. Only the center patch, which consists of two colors, are used in node identification. The *location* of a testbed node is the centroid of the ID patch. The *orientation* is determined based on its direction, computed as the vector connecting the centroid of the tail patch to that of the head patch. Using same colors for head and tail patches introduces redundancies that guard against noises and simplifies the determination of robot orientation.

The color recognition algorithm used in MiNT-m is extremely simple and thus efficient, and uses standard image processing techniques for edge detection. It scans the pixels in a captured image one by one. If the algorithm detects a pixel of a known color, it searches the neighboring pixels to check if they are of the same color, and grows pixels of the same color into regions as much as possible. Combinations of neighboring color patches are used to identify individual testbed nodes.

Factors that affect the accuracy of MiNT-m's color-based position/orientation tracking algorithm are the size of each color patch, the number of distinct colors used, the stability of lighting condition, optical noise in the patch boundaries, which might distort centroid computation. Given the patch size and the camera resolution used in the current MiNT-m prototype, a 1-pixel recognition error could potentially translate to 0.27 inch in location error, and 2.2 degrees in orientation. As an optimization, it is possible to exploit inter-frame coherency to greatly mitigate the effects of recognition errors.

4.3 Node Trajectory Determination

MiNT-m's trajectory computation is based on a static trajectory planning algorithm, which computes a robot's path assuming the world is static, and a dynamic collision avoidance algorithm, which detects and resolves collision by fine-tuning pre-computed trajectories.

Algorithm 1 Node Trajectory Determination

```
for (∀ nodes marked for mobility) do
  obstacle ← Nearest obstacle on direct path between A_initial and A_final
  if (obstacle == 0) then
    // There is a direct path to the destination
    Generate Roomba moves
  else
    Determine intermediate points (P_1, P_2, ..., P_n) on lines ⊥ or at angle θ to direct path passing through the nearest obstacle;
    Check for direct path between A_initial and any of (P_1, P_2, ..., P_n) ;
    Check for direct path between any of (P_1, P_2, ..., P_n) and A_final ;
    if (∃ 2-hop direct path via P_i) then
      Generate Roomba moves from A_initial to P_i ;
      Generate Roomba moves from P_i to A_final ;
    else
      if (∃ direct path from A_initial to some P_i) then
        Generate Roomba moves from A_initial to P_i ;
      else
        Move δ steps in a random direction away from nearest obstacle;
      end if
    end if
  end if
end for
```

Given the current position and the target destination of a testbed node (TN), the control server takes a snapshot of the positions of other testbed nodes and treats them as obstacles in the calculation of the TN's trajectory. The static trajectory planning algorithm first checks if there is a direct path between the TN's current position and its destination. If such path does not exist, the algorithm identifies the obstacle closest to the source position, and finds a set of intermediate points that lie on the line which passes through the obstacle and is perpendicular to the line adjoining the source and destination and have a direct path to both the source and destination. If no such intermediate points exist, the algorithm finds a random intermediate point that is δ steps away from the obstacle closest to the source and is directly connected to the source, and repeats the algorithm from this new intermediate point as if it is a new source. The algorithm is shown in Algorithm 1.

In Figure 5, node N_1 is set to move from $A_{initial}$ to A_{final}. However, N_2, N_3 and N_4 block the direct path between $A_{initial}$ and A_{final}. The trajectory planning algorithm first figures out that N_3 is the obstacle closest to $A_{initial}$, and then computes the intermediate points $P_1, P_2, ..., P_6$ to search for 2-hop paths to A_{final}. Because the paths L_1 and L_2 are partially blocked, the algorithm eventually chooses path L_3, which passes through the intermediate point P_3.

In addition to static trajectory planning, MiNT-m also requires a dynamic collision avoidance algorithm because testbed nodes could be moving and the robot movement is not perfect. Given a snapshot of the testbed, which appears once every 1/4 second in the current prototype, MiNT-m performs a proximity check for each testbed node. If any two nodes are closer than a threshold distance, both of them

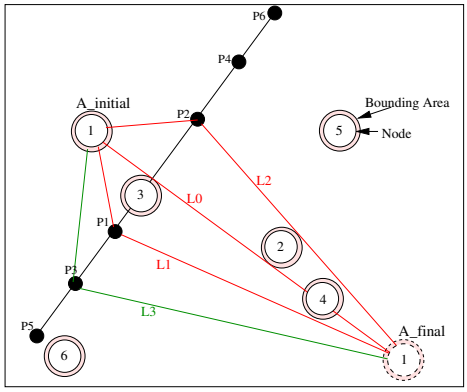

Figure 5: *Finding the trajectory from N1's current position $A_{initial}$ to A_{final}. As nodes N2, N3 and N4 block the direct path, the algorithm tries to identify an alternate 2-hop path to move N1 from its current source to its destination.*

stop, a new path is re-computed for each of them, and the algorithm moves them on their new trajectory one by one. In the event that two nodes indeed collide with each other, the algorithm again detects it through a proximity check and stops the nodes immediately. In this case, the algorithm also recomputes a new path for each of the two nodes, and moves them one by one.

One problem with Roomba is that its movement is not very accurate, which could also lead to dynamic collision. Its forward movement is 5 inches per step, and the pivot movement varies from 4 to 5 degrees per step. Furthermore, a Roomba only allows three types of movement: forward, clockwise turn and counter-clockwise turn. Hence backward movement is implemented by a turn of 180 degree followed by forward motion. The inaccuracies in Roomba movement necessitate constant correction. Additionally, MiNT-m's object tracking errors also contribute to position inaccuracy. Hence, after each sequence of 5 move commands are applied to a testbed node, its trajectory is re-computed till it reaches the destination point.

5. 24X7 AUTONOMOUS OPERATIONS

A key challenge in the design of MiNT-m is how to render the testbed self-manageable and providing uninterrupted 24x7 continuous operation. Since each testbed node is battery powered, the batteries must be recharged periodically. Usually battery charging is a manual process that requires the administrator to take discharged nodes to charging stations [12]. In contrast, MiNT-m supports automatic recharging of the nodes' batteries and imposes zero manual charging overhead. In this section, we present the auto-recharging mechanism, the residual battery capacity estimation mechanism, and the recharge scheduling policy. Finally, we discuss how we recover from node crashes resulting from bugs in protocols under test.

5.1 Auto-Recharging Mechanism

Roomba provides a docking station to charge its batteries. The Roomba docking station emits an IR beacon that is received by a Roomba over a distance of around 5 ft. When a Roomba's battery power drops below a threshold, it starts looking for a beacon emitted by the docking station and uses the signal to home into the docking station for recharge. Unfortunately, Roomba's built-in battery cannot be used to directly power the RouterBOARD. Hence, we use a separate universal laptop battery to power the RouterBOARD. To recharge the RouterBOARD battery along with the Roomba battery, we connect the RouterBOARD battery to the charging tip of the Roomba battery as shown in Figure 6. This allows both the batteries to be charged simultaneously from the same docking station.

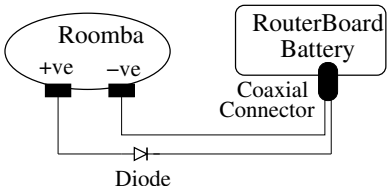

Figure 6: *The auto-charging circuit for charging the wireless node battery when the mobile node docks itself into the docking station for recharging.*

5.2 Residual Charge Estimation

To keep the testbed running on a 24x7 basis, nodes with low residual charge must be scheduled for recharge on every charging cycle. Unfortunately, neither the Roomba nor the RouterBOARD battery provide any API for probing the residual battery capacity. Hence the residual charge on a testbed node is estimated based on profiling of the batteries and the node's usage.

For each node, the residual charge on the Roomba's internal battery and the RouterBOARD's battery are estimated separately. The residual charge on Roomba's internal battery is estimated using the equation:

$$R_{roomba} = I_{roomba} - N_{roomba} * U_{roomba} \quad (1)$$

where R_{roomba} and I_{roomba} are the residual and the initial charge on Roomba's battery respectively. N_{roomba} is the number of movements performed by Roomba, and U_{roomba} is the energy consumed per movement. Although, a Roomba battery drains even when Roomba is not moving, this is negligible.

Similarly, the residual charge on RouterBOARD's battery is estimated as:

$$R_{board} = I_{board} - T_{board} * U_{board} - N_{disk} * U_{disk} - N_{packet} * U_{packet} \quad (2)$$

Here, R_{board} and I_{board} are the residual and the initial charge on RouterBOARD's battery respectively. T_{board} is the amount of time RouterBOARD has been on, and U_{board} is its idle power consumption per unit time. N_{disk} and U_{disk} are the number of hard disk operations performed by the RouterBOARD and the energy consumed per disk operation respectively. Finally, N_{packet} and U_{packet} are the number of packets sent/received by the RouterBOARD and the energy consumed per network operation respectively. The energy consumed by each IR transmission is negligible.

5.3 Node Re-charge Scheduling Algorithm

A node cannot be used for experimentation while it is being charged. However for 24x7 operation a set of nodes must be operational at all times. Hence, a set of spare nodes are provisioned in the testbed. Specifically, the testbed uses $n + m$ nodes, where n nodes are used in the experiments at any time while m nodes are being recharged. If c is the average charging time for a node and d is its average discharging time, then by maintaining $(m > n * c/d)$ spare

Figure 7: *The webcam shot of the testbed. The color patch on each node uniquely identifies the node as well as tells its position and orientation.*

nodes, the testbed can be run without any downtime. Neither the Roomba's nor the RouterBOARD's battery suffer from any memory effect due to incomplete charge/discharge cycles. Therefore, the actual recharge scheduling algorithm is straightforward: at every recharge cycle, dock the m least charged nodes out of $n + m$ testbed nodes for recharge.

5.4 Node Crash Recovery

Another aspect of 24x7 operation is dealing with node crash due to bugs in kernel modules. We utilize the 2 hardware watchdog controllers that each RouterBOARD is equipped with. A software daemon periodically writes some bytes to an I/O port indicating to the watchdog controller that the node is alive. In the event of a node crash, the control daemon stops writing to the I/O port, and the control server detects it after a timeout and automatically reboots the node. To ensure that a reboot restores the original kernel image, no user-specified module is loaded at boot-time. This ensures that a crashed node can always automatically recover within a fixed time.

6. MOVIE: MINT-M CONTROL AND VISUALIZATION INTERFACE

The graphical user interface for MiNT-m is derived from Network Animator (NAM) [2], that is one of the most commonly used visualization tools for ns-2. NAM is a Tcl/TK based animation tool designed for offline viewing of network simulation traces. It features topology layout, packet level animation, and various data inspection tools. The input to NAM is a list of $< attribute, value >$ pairs. Based on the *attribute* type, NAM displays different objects, like nodes with node id and links with link characteristics.

An interface for management and control of a wireless testbed benefits with more interactive features. The user must be able to control node movements as well as configure various node-level parameters. Also, the interface must provide real-time status update of different objects, like nodes, links, routes in the testbed. Finally, during experimentation, important events in the traces should be displayed in real time, instead of offline. To incorporate these capabilities, NAM is evolved into an interface for MiNT-m, called Mint-m cOntrol and Visualization InterfacE (MOVIE) (Fig 8). This section presents the features of MOVIE along with implementation details of some of its advanced features.

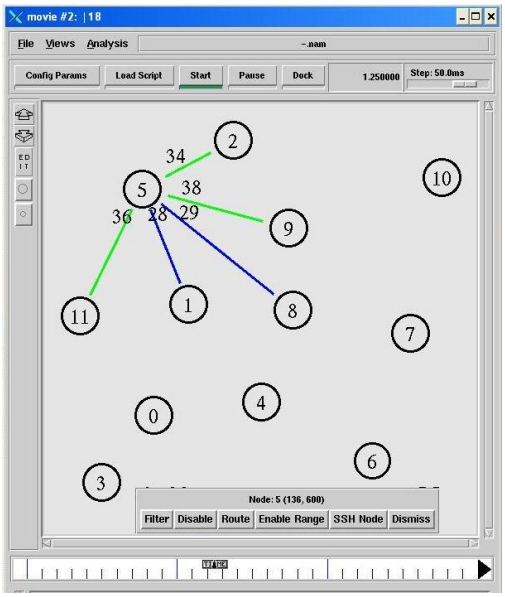

Figure 8: *MOVIE GUI acts as the front-end to the testbed, and supports all management, control, and visualization functionalities. Each node icon represents the actual position of a physical node in the testbed (as shown in Figure 7). Nodes are physically moved by dragging the corresponding icons in the GUI. The number on each link represent the signal quality for the link in that direction. MOVIE can be used to set the network-wide parameters, and override them on a per-node basis.*

6.1 MOVIE Visualization Features

Based on the periodic updates from testbed nodes, MOVIE displays states of different objects such as *nodes, links, and routes*. As different events are triggered on the testbed, they are shown in MOVIE in real time. However, displaying each and every packet exchange substantially increases the control traffic, as well as hides useful information. Therefore we extract only the important events such as route changes, link quality changes, node movements, and display them in real time. Since MOVIE displays all these events and states as they occur, the events generated by different nodes need to be merged to produce a unified sorted list of events, a job performed by the control daemon. We now discuss the exact testbed attributes displayed in MOVIE, and how they are extracted before being sent to the control daemon.

Node Attributes: Each testbed node is represented by an icon in MOVIE: The position and orientation of the icon correspond to that of the node in the testbed arena. Double-clicking on a node icon opens a window that displays various node-level attributes. These include network card configuration, such as MAC address, radio channel, and BSSID. The window further displays the residual charge on the node batteries as estimated by the algorithm discussed in Section 5.2. Right-clicking on the node icon displays its hearing range neighbors, while left-clicking on it displays the node's interference-range neighbors. The determination of interference-range neighbors is discussed in following subsection.

Link Attributes: MOVIE also displays link-level characteristics, such as signal quality, error rate, and traffic load on the link. All these characteristics are measured in a passive

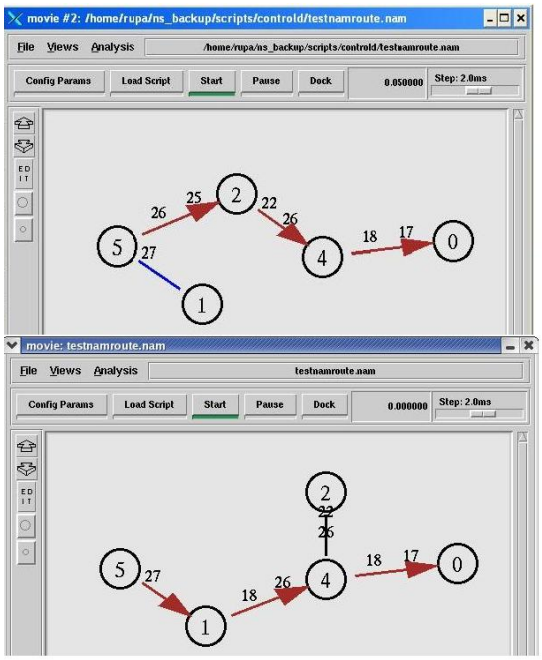

Figure 9: *The changes in multi-hop routes are displayed in real time in MOVIE. In the figure, the route switch from node 2 to node 1 is sent to the control node, and displayed in MOVIE in real time.*

manner, upon reception of every wireless packet. To measure the link error rate, each packet is stamped by a unique monotonically increasing sequence number.

Multi-hop Routes: Protocol debugging can be much simplified through visualization of multi-hop routes discovered by the routing protocol (like AODV) in use. Each node maintains all the route table updates and periodically sends them back to the controller node. Changes in routes are also displayed in real time through MOVIE, as shown in Fig 9.

Protocol-specific Attribute Display: During development of new protocols, it is often necessary to view the changing values of different protocol-specific attributes, such as TCP's congestion window. MOVIE can display arbitrary attribute values as long as they are exported by the protocol developer as attribute-value pairs. For ns-2 based protocols, the attribute can be simply exported as part of the NAM file. For real implementations, we use the proc filesystem interface to export various attributes. For new protocols that plan to use the MiNT-m infrastructure, the values can be directly written in an attribute-value format. For old protocols, which do not write in the desired format, a parser must be installed that translates the results from the files into the attribute-value format. These values are then sent for display to MOVIE.

Determination of Communication and Interference Neighbors: While setting up a topology for a wireless experiment, an important step is to understand the interference relationship among the neighbors. MOVIE can highlight both the communication and sense range neighbors for any specified node.

Determination of hearing range neighbors is straightforward: The chosen node sends out a broadcast ping. All the neighbors that respond to the ping request are the hearing range neighbors of the chosen node.

Determining interference range neighbors is relatively trickier: A node may not normally hear the transmissions from its interference range neighbors, but can sense their transmission. For 802.11a transmissions at 6 Mbps, if the transmit power of a card is increased by 2 dBm, then the sense range neighbors become hearing range neighbors. Hence, in order to determine sense range neighbors of a node, the transmission rate is set to 6 Mbps, wireless card transmit power is increased by 2dBm, and then a broadcast ping is sent. The nodes that respond are the sense range neighbors for this node at the default transmission rate and transmit power.

6.2 MOVIE Control Features

MOVIE front-end (Figure 8) is designed to allow users to configure the testbed on an experiment-by-experiment basis. It provides all the necessary controls, collects detailed information from the testbed, and gives real-time status update to the users. Control activities of a user generate downstream data flow from MOVIE to the nodes. Visualization functions feed data into MOVIE for display. This subsection discusses the main control features supported by MOVIE.

Node Configuration: For each testbed node, the user can set various configuration parameters such as card transmit power, retry count, sensitivity threshold, and RTS threshold. Most of these parameters are set using standard wireless card API. The sensitivity threshold is the only one that is set by directly altering a card register.

Topology Configuration: MOVIE allows user to position the nodes at desired locations in the testbed by dragging the corresponding icons in the GUI. The movement of a node icon generates a destination point and triggers trajectory computation on the controller node (discussed in Section 4.3). The controller node then issues *move* commands to the node daemons on the corresponding nodes.

Given a certain placement of nodes, the node density can be altered by changing the transmit power level of the nodes. Further fine-tuning of topology is possible by selectively disabling individual links and routes. Links are disabled through use of MAC filtering function that drops all packets going from a specified source to a destination. Route disabling is done by periodically deleting the corresponding route table entry from all the nodes along the path.

Pause/Breakpointing in Hybrid Simulation: In hybrid simulation mode, the simulator is running in a distributed manner across all the nodes in the testbed. Debugging such a distributed application is a challenging task. In addition to simultaneous start and stop of an experiment on all nodes, MiNT-m simplifies protocol debugging by introducing other standard features of a typical debugger, namely pause and breakpointing of the experiment.

The implementation of the *pause* feature in MiNT-m requires modification to the RealTime scheduler in the hybrid ns-2. Normally, the real-time scheduler sets the simulator's clock value to the system clock. To account for pause, the total pause period is measured and subtracted from the system clock to update the simulator's clock. When the simulation is paused, the execution of events pending in the event queue as well as those in transit to other nodes, is stalled. However, since the simulator's clock is also paused, no adjustment is needed to the time for the events in the event

queue. In the pause state, the user is allowed to change the physical configuration of the testbed, or alter any physical parameters of the nodes in the testbed, like node positions or transmit power, before resuming the execution.

The *breakpoint* feature is implemented by using the pause mechanism. In breakpointing, the user specifies a watch on ns-2 packet header fields. Each node matches the outgoing/incoming packet headers for pre-specified values, and when a match occurs, a breakpoint signal is sent to the controller node. The controller node then informs all the nodes to pause their experiment execution.

Rollback Execution in Hybrid Simulation: The *rollback* feature for an experiment running in hybrid simulation mode gives the flexibility to a user to repeat the experiment from a snapshot time in the past with modified parameters fed to the experiment. This saves on experimentation time as the entire simulation experiment need not be repeated from the beginning.

In order to implement this feature, the state of the executing process (hybrid ns-2 in case of MiNT-m) is stored at regular intervals. On a rollback request, the saved state is loaded and execution repeats from that point. The controller node triggers each node controller to fork the ns-2 processes running on a node and stores the process id of the forked process. On rollback, the stored process closest in time to the rollback time is selected. The remaining stored processes later in time to the executing process are purged from the stored process list. In order to maintain consistency of the display, once the node controllers report that the rollback operation has succeeded, then MOVIE also reverts all events it has processed till the rollback time by looking up the history. Once both MOVIE and each node controller have successfully completed the rollback initiation phase, the experiment is restarted from the user interface.

Note that this feature also rolls back the node positions and the ns-2 script execution. It is, however, not possible to rollback channel conditions.

7. MINT-M EVALUATION

In this section, we present an evaluation of different components of MINT-m, namely the tracking system, the collision avoidance algorithm, the auto-charging mechanism, and the hybrid ns-2 simulator. We also present the evaluation of a state-of-the-art MANET transport protocol ATP on the testbed using the MiNT-m's hybrid simulation mode. A brief cost evaluation of the 12-node testbed is also presented. Finally, lessons learnt from using MiNT-m remotely is discussed.

7.1 Tracking Accuracy and Scalability

The tracking system is required to get accurate position and orientation information of each node in the testbed. Ideally, if each step of the Roomba movement is exact, then it is possible to figure out the current location of the node, based on the initial position and the steps executed. However, due to floor friction and mechanical non-homogeneity of the Roombas, the Roomba movement is imprecise. Figure 10 shows the inconsistency in Roomba movements for both the *move forward* as well as the *rotate* commands. This makes it necessary to design a full-scale vision based tracking system that can provide a more accurate location information than that based on odometry.

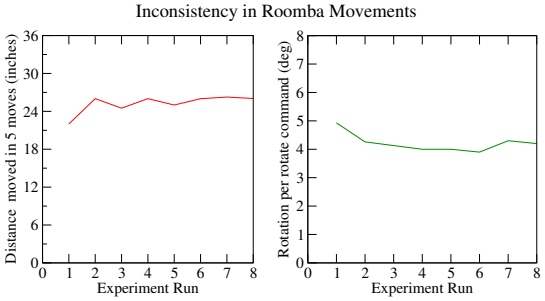

Figure 10: *Roomba movement with each set of move/rotate commands. The distance traveled and rotation performed are not consistent making it difficult to track the Roomba position based on steps executed. This inconsistency makes the vision-based tracking system indispensable to keep track of current node positions.*

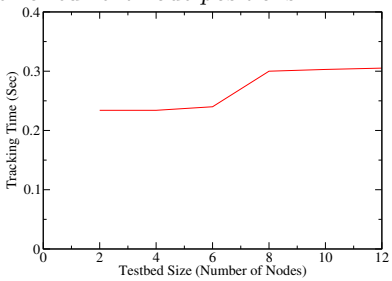

Figure 11: *Scalability of the tracking system in terms of nodes tracked. With increasing number of nodes the the time to locate each node increases. With 12 nodes, the tracking system can produce one location update every 0.3 sec.*

We measured the *locationing accuracy* of the tracking system. The inaccuracy is measured as the difference between the location and orientation of a MiNT-m node as returned by the tracking system and its true coordinates. The mean error in the coordinate distance is 0.95 inches with a standard deviation of 1.17 inches. The mean error in orientation is 3.36 deg with a standard deviation of 2.77 deg.

Another important factor is the *scalability* of MiNT-m's object tracking algorithm with increasing number of nodes. This is measured as the time taken for end-to-end tracking (including frame grabbing, node identification, node locationing, and merging of location data from multiple tracking servers) as the number of nodes in the testbed increases. In Figure 11, we plot the time taken by the tracking server to produce one set of node locations. Although the tracking is done in parallel on all the tracking servers, multiple nodes could be clustered within the area covered by one tracking server. This leads to an initial increase in the tracking overhead with number of nodes. With 12 nodes, the tracking system could produce one location update for all the nodes every 300 msec. The maximum number of nodes that can fall within the coverage area of a tracking server is limited. Hence, as the testbed scales further and more tracking servers are added, the tracking overhead should not increase any further.

One of the physical constraints we had was the height of the ceiling. If the webcams could be placed higher up, then each webcam could cover a larger area thus scaling the testbed size. To evaluate this theory, we scaled down the size of the images we got from the webcams, and ran the

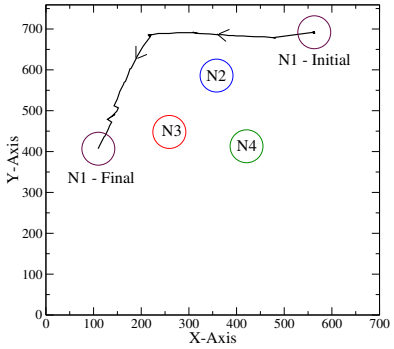

Figure 12: *Node path trace collected from MOVIE: Given a destination a node can detect the presence of obstacles, and find a collision free path.*

Number of Nodes	Reconfiguration Time (sec)
2	18
3	38
4	47
5	82
6	100

Table 1: *Topology re-configuration time increases with the size of the testbed.*

tracking algorithm on them. Even when each webcam image is shrunk to 1/16th of its original size (equivalent to placing the camera at 4 times the current height), the tracking system works well. In particular, the tracking system's locationing error just increased from 0.95 inches to 2.32 inches, while the error in reported orientation increased from 3.36 deg to 4.11 deg.

7.2 Collision Avoidance

Figure 12 shows the path followed by a node as a result of the trajectory determination algorithm used in MiNT-m. There are three other obstacle nodes on the mobile node's path. The trajectory determination algorithm takes the obstacles into account, and generates a path that avoids these obstacles. The line in the figure shows the resulting path.

To get an estimate of the overhead introduced by the combination of position/orientation tracking and collision avoidance algorithm, we measured the time taken by a mobile node to move from initial marked position to final marked position once with (i) tracking system and collision avoidance turned *on*, and next with (ii) tracking system and collision avoidance turned *off* and the mobile node moving through the same path selected in case-(i). The time taken for case-(i) was 31 sec, while that for case-(ii) was 26 sec, showing that tracking system and collision avoidance algorithm combined induce a 20% overhead on configuration time.

Table 1 presents the topology reconfiguration time. In each experiment run, all nodes started in parallel from fixed initial positions around the corner of the testbed arena. The final position of each node was chosen randomly and kept constant for all the experiments. As more nodes are introduced and they try to reach their destination in parallel, there are effectively more dynamic obstacles present in the environment leading to increase in the time to reach

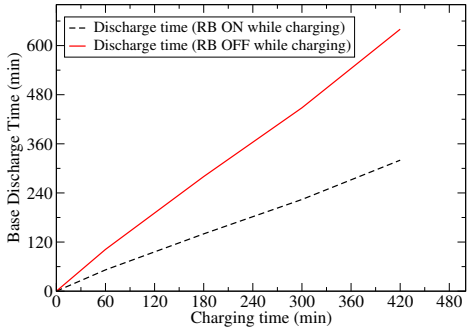

Figure 13: *Charge and discharge cycles of a MINT-m node. If the RouterBOARD is halted during charging, the battery charges faster, hence it runs longer as shown by the increase in discharge time.*

the final topology configuration. Most current testbeds either do not support experiment-by-experiment topology reconfiguration, or require several hours to come up with a specified topology. Comparatively, MiNT-m reconfiguration takes time of the order of minutes.

7.3 Auto Re-charging

We measured the charge and discharge times for the two batteries. This information is used by the residual charge estimation algorithm to predict when a particular node needs to be recharged. With a fully charged battery, the RouterBOARD lasts around 13.5 hours without performing any network or hard disk operations. The runtime reduction due to different activities are: 2.05 sec for every 1M network operations, and 8.82 sec for 1K disk operations. The runtime reduction due to IR operations is negligible. On the other hand, the Roomba battery lasts for 2 weeks without movement, and can perform 13840 moves till the battery dies. Since the number of mobility commands executed are less than the RouterBOARD activities, therefore the RouterBOARD usually depletes faster. The full charging time for the Roomba battery is also 3 hours, which is less than the time to charge the RouterBOARD's battery.

Figure 13 shows the base discharge time (no network card or hard disk activity) for the RouterBOARD battery when the node has been charged for different periods of time. The linearity of the discharge time with respect to charge time simplifies the algorithm used to estimate how much charge the battery has accumulated for a certain charging duration.

If the RouterBOARD is active during the charging process, the battery gets depleted while charging. This leads to a faster discharge during operation. To increase the lifetime, the RouterBOARD is put into a *halt* state during charging, and powered up using Wake-on-wireless LAN feature available on the wireless network cards before putting it back into operation. This technique produces a substantial improvement on a testbed node's battery lifetime, as shown in Figure 13.

7.4 Hybrid-ns Performance

The core computing platform we use is a RouterBOARD-230 that has a 266 MHz CPU and is processor-limited. As we add more processing overhead on the system, the maximum throughput we can achieve goes down. We measured the throughput degradation of a single hop as we enable different features: remote tracing, per-packet local tracing,

Tracing Type	Throughput (Mbps)
No Tracing	20.51
On-line Remote Tracing	17.043
Per-packet Application Tracing	8.423
Per-packet Router Tracing	7.83
Per-packet MAC Tracing	16.08
Per-packet Full Tracing	5.53

Table 2: *Hybrid-ns throughput as the tracing is turned on. Due to tracing overhead the available throughput drops.*

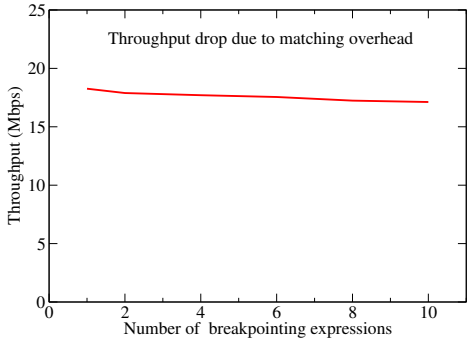

Figure 14: *This graph shows that the throughput of hybrid simulation will drop as expressions are added to the breakpoint list.*

experiment breakpointing, and experiment rollback.

We first look at the impact of different forms of tracing on the maximum throughput achieved between two communicating MiNT-m nodes. Table 2 lists the results. Interestingly, even without tracing, ns-2 application agents could only achieve 20.5 Mbps as compared to 33 Mbps achievable by a simple UDP flow running between the same nodes. This is because of the additional processing overhead introduced by ns-2, in contrast with a simple UDP sender that needs almost no processing to prepare a packet. To achieve 20.5 Mbps throughput, we did few optimizations to ns-2 such as use of heap scheduler instead of dynamic hash-based scheduler. This was required to avoid stalling of ns-2 during the frequent re-hash operation done by the hash scheduler.

Any form of tracing introduces further CPU processing overhead due to string operations done by ns-2. The on-line remote tracing is done only for selected events and hence results in least degradation (3.5 Mbps). Per-packet tracing introduces maximum overhead. But even with full per-packet local tracing and on-line remote tracing turned on, the nodes could achieve 5.53 Mbps, enough to saturate a channel operating at 6 Mbps link rate. Current hybrid-ns implementation copies entire protocol packet including its payload between user-space and kernel-space for sending/receiving. As simulated protocols do not care about the payload, one potential optimization is to only copy protocol headers.

Another feature of hybrid-ns is *breakpointing* of experiments. This feature requires matching expressions to trigger the breakpoints when the event occurs. Since matching different fields incurs overhead, the throughput reduces. In Figure 14, we show the impact of increasing number of breakpoint expressions on the throughput of hybrid simulation. Despite the CPU bottleneck, the overhead increases only slightly with increasing number of expressions. This is because the expressions are only checked once for each packet, limiting the extra processing burden introduced by breakpointing.

Item	Cost ($)
Wireless Node	
RouterBOARD RB-230	330
Wireless NICs and antennas	100x4 = 400
MiniPCI Adapter	65
Attenuators	40x6 = 240
Hard Disk	75
External Laptop Battery	170
Spitfire	135
Roomba	250
Total	**1665**
Tracking Server	
Desktop PC	300x3=900
Quickcam 4000	100x6=600
Total	**1500**
Control Server	
Wireless NICs and antennas	100x3 = 300
Desktop PC	300
Total	**600**

Table 3: *Cost breakup of MiNT-m infrastructure.*

We similarly evaluated the performance of *rollback* feature. This feature requires regular snapshot (using fork() system call) of ns-2 process running on every MiNT-m node. Linux kernel's fork() system call automatically uses copy-on-write technique to avoid copying of all the pages at the fork time. This spreads out the throughput degradation to a few seconds after the fork() system call. With even a 1 minute snapshot granularity, the overall throughput degradation was less than 0.25 Mbps.

7.5 Cost Evaluation

A major design goal for MiNT-m is to develop an inexpensive mobile wireless network testbed that is built from commercial off-the-shelf components and can eventually be duplicated in other institutions. None of the components in MiNT-m are custom made, hence it is relatively straightforward to replicate the testbed elsewhere. Table 3 shows the cost break-down of the components used in constructing a single MiNT-m node, as well as the cost of the tracking server and the control server. The current 12-node MiNT-m prototype is implemented at an overall cost of around $22,000.

7.6 Protocol Evaluation on MiNT-m

To demonstrate the usefulness of MiNT-m as an experimentation platform, we studied a cross-layer MANET transport protocol called ATP [19] using the MiNT-m's hybrid-ns simulation feature. Although, ATP has been comprehensively evaluated in pure ns-2 simulations, its behavior has not been studied on a real testbed. Our experimental study revealed several important characteristics of the protocol that we discuss in this subsection.

7.6.1 ATP Overview

In ATP scheme, every intermediate node measures queuing and transmission delays for each packet passing through it. The sum of exponentially averaged queuing and transmission delays yields the *average packet service time* experienced by all the flows going through the intermediate node.

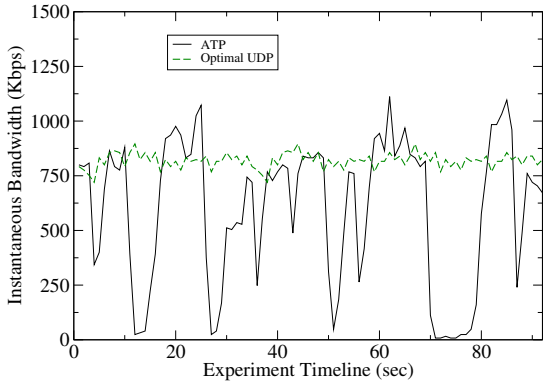

Figure 15: *Fluctuations in ATP's bandwidth estimation. The channel bandwidth is fairly constant, as seen by an optimal UDP stream sent over the same 3 hops.*

In the equilibrium condition, each ATP flow attempts to maintain exactly one data packet on every router along the path [19]. The service time therefore reflects the ideal dispatch interval for all the flows competing over the bottleneck link. Every packet bears the maximum service time encountered on any of the intermediate hops. The bottleneck service time is communicated back to the sender, which adjusts its packet dispatch interval to match this service time.

7.6.2 ATP's Inaccurate Bandwidth Estimation

One of the key issues we observed with ATP is bandwidth under-estimation. Figure 15 shows the fluctuations in bandwidth estimation by an ATP flow originator performing FTP upload to a node that is 3 hops away. The same figure also shows the optimal bandwidth as seen by a UDP flow. The optimal flow was found by sending a UDP stream at different rates until the maximum was achieved. The stability of the optimal UDP flow suggests that the channel bandwidth fluctuations are negligible, and most of the bandwidth fluctuations are internal to the ATP protocol itself.

Further experimentation revealed the root of the problem to be the service time measurement metric proposed by ATP. The problem with the overall-service-time approach is that it couples the queue-size management with rate estimation, which leads to traffic fluctuations and in turn non-optimal estimation of channel bandwidth [17]. More concretely, it is very hard to maintain exactly one data packet from each flow on every router. If there is even a slight change in transmission time of a single packet, a queue (say of two packets) builds up on the router for the rest of the epoch. Clearly, an extra packet in the queue does not indicate any change in the network bandwidth. However, in the next epoch ATP sender proportionally reduces its sending rate (to half in this example) to bring the queue down to one packet. It is in these epochs, that an ATP sender under-estimates the path bandwidth.

7.6.3 ATP's Flow Unfairness

We tested several of the scenarios and came up with two common ones where ATP demonstrates substantial unfairness. These scenarios are depicted in Figure 16. The first example (Fig 16 (a)) corresponds to the hidden terminal scenario. Here one wireless link's transmission is inhibited by another link, eventually leading to unequal bandwidth allocation between the two. Table 4 shows the resulting

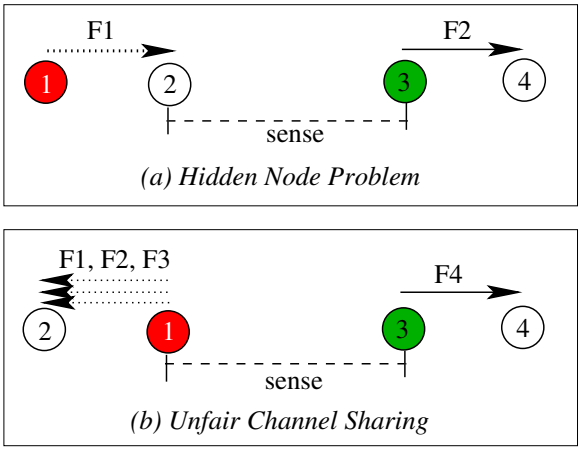

Figure 16: *ATP's unfairness scenarios: The wireless node getting a lesser than fair share of bandwidth is numbered 1 (and colored in red or white), whereas the one getting a larger share is numbered 3 (and colored in green or black). (a) Node 1 lacks information about Node 3's transmissions, attempts its communication at inopportune times, and eventually backs off unnecessarily. (b) Flow F1, F2, F3, and F4 all share the same channel, but ATP, like most other transport protocols, allocates more bandwidth to F4 than to others.*

Flow	ATP Thruput (Kbps)	Optimal Thruput (Kbps)
Hidden (Flow 1)	570.4	985.8
Inhibitor (Flow 2)	1104.6	1186.4

Table 4: *ATP accentuates the hidden terminal problem.*

bandwidth distribution between the two flows. Flow 1 originating from the hidden node gets much lesser bandwidth than Flow 2 originating from the inhibiting node. Although one could attribute this problem solely to the 802.11 MAC layer, this problem can indeed be addressed at the transport layer as demonstrated by fairness of optimal UDP flows going over the same network.

The second example (Fig 16 (b)) corresponds to a general channel space sharing scenario. Concretely, ATP allocates a radio channel's bandwidth fairly among flows from a single node, rather than among all flows from all nodes that share the radio channel. As a result, a flow emanating from a node with fewer flows tends to get a larger than fair share of channel bandwidth. This is shown in Table 5 where Flow 4 gets much larger than its fair share, while Flow 1 and Flow 3 suffer.

7.6.4 Discussion

This protocol study demonstrates the usefulness of different MiNT-m features. Specifically, the ability to perform

Flow Id	ATP Thruput (Kbps)	Optimal Thruput (Kbps)
1	383.0	641.6
2	590.1	641.6
3	484.7	641.6
4	1010.8	641.6

Table 5: *ATP's fairness in typical channel sharing scenario.*

hybrid-ns simulations enabled us to re-use the pure ns-2 code written for ATP. Similarly, the ability to reconfigure topology through MOVIE enabled us to come up with specific topologies that revealed the protocol weaknesses. MOVIE enabled us to visualize the queue size on every ATP router, thus helped pinpointing the reason behind ATP's fluctuations.

8. CONCLUSION

Though there are several on-going efforts that aim to construct high-fidelity general-purpose wireless network testbeds, none of them provide adequate support for network topology reconfiguration and node mobility. While mobile robot technology appears to be an obvious solution to this problem, the associated engineering challenges are quite formidable because of the following conflicting requirements. First, the mobile robots used have to be low-cost because we need a large number of them. Second, the mobile robots need to be completely untethered so that unconstrained physical node movement is possible. Third, the amount of human intervention required to manage the mobile robots must be minimal as they are meant to be the building blocks of a research infrastructure operating at 24 hours per day and 7 days per week. In this paper, we describe how MiNT-m, a generic experimentation platform for mobile multi-hop wireless network protocols, resolves these issues and strikes a good engineering balance among them. MiNT-m shrinks the amount of physical space required for experimenting with multi-hop wireless network protocols through radio signal attenuation. In addition, MiNT-m features several unique innovations:

- MiNT-m turns a consumer-grade robotic vacuum cleaner appliance called Roomba into a general-purpose mobile robot platform whose movement can be wirelessly controlled from a central server.
- To support network topology reconfiguration and node mobility, MiNT-m features a vision-based robot tracking system that requires only commodity web cameras, and a dynamic collision avoidance algorithm for the planning of node movement trajectories.
- To support fully autonomous 24x7 operation, MiNT-m includes an automatic battery recharging facility that constantly predicts the battery usage of individual robots and schedules them for recharging when their battery runs low.
- MiNT-m's network management interface significantly improves protocol simulation productivity because it offers users full interactive control of simulation runs and real-time visualization of simulated protocols' states and testbed network conditions. In particular, it supports advanced debugging features such as protocol-specific breakpointing and reverse protocol simulation.

Although MiNT-m was originally designed to support wireless network emulation, its scalable robot positioning and movement control, automatic self-recharging, comprehensive monitoring and control, make it an effective platform for general collaborative robot computing research as well, e.g., developing automatic guided vehicles for transporting goods within factories or warehouses. The major future work of the MiNT-m project is to scale the testbed from the current 12-node configuration to a 100-node configuration, including its optical object tracking system.

9. ACKNOWLEDGMENTS

Thanks to all the people at Stony Brook University who have helped in the MiNT project. Special thanks to our shepherd, Rahul Sukthankar and all the reviewers for their useful comments. This research is supported by NSF award CNS0435373, NYS Millennium Center grant, and New York State Center for Excellence in Wireless and Information Technologies (CEWIT) at Stony Brook University.

10. REFERENCES

[1] Roomba Discovery – http://www.irobot.com/
[2] The Network Simulator (ns-2) – http://www.isi.edu/nsnam/ns/
[3] Universal Infrared Remote Control From Any PC – http://www.innotechsystems.com/spitfire6001.htm
[4] Ashish Raniwala and Tzi-cker Chiueh. Architecture and Algorithms for an IEEE 802.11-based Multi-channel Wireless Mesh Network. In *Proc. of IEEE Infocom*, 2005.
[5] B. A. Chambers. The Grid Roofnet: A Rooftop Ad Hoc Wireless Network. Technical report, MIT Master's Thesis, Jun 2002.
[6] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.
[7] L. Cremean, W. Dunbar, D. van Gogh, J. Hickey, E. Klavins, J. Meltzer, and R. M. Murray. The Caltech Multi-Vehicle Wireless Testbed. In *Proc. of Conference on Decision and Control*, 2002.
[8] P. De, A. Neogi, and T. Chiueh. VirtualWire : A Fault Injection and Analysis Tool for Network Protocols. In *Proceedings of International Conference on Distributed Computing Systems*, May 2003.
[9] P. De, A. Raniwala, S. Sharma, and T. cker Chiueh. MiNT: A Miniaturized Network Testbed for Mobile Wireless Research. In *Proc. of IEEE Infocom*, 2005.
[10] S. Graham and P. R. Kumar. The Convergence of Control, Communication, and Computation. In *Proceedings of Personal Wireless Communication (PWC)*, 2003.
[11] J. Heidemann, N. Bulusu, and J. Elson. Effects of Detail in Wireless Network Simulation. In *Proc. of the SCS Multiconference on Distributed Simulation*, January 2001.
[12] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau. Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. In *IEEE Infocom*, 2006.
[13] S. Kurkowski, T. Camp, and M. Colagrosso. A Visualization and Animation Tool for NS-2 Wireless Simulations: iNSpect. In *Technical Report MCS-04-03, The Colorado School of Mines*, 2004.
[14] H. Lunndgren, D. Lundberg, J. Nielsen, E. Nordstrom, and C. Tscudin. A Large-scale Testbed for Reproducible Ad Hoc Protocol Evaluations. In *Proc. of WCNC*, 2002.
[15] D. Maltz, J. Broch, and D. Johnson. Experiences Designing and Building a Multi-Hop Wireless Ad-Hoc Network Testbed. In *CMU TR99-116*, 1999.
[16] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket location-support system. In *Proc. of International Conference on Mobile Computing and Networking*, 2000.
[17] S. K. Raj Jain and R. Viswanatha. Rate Based Schemes: Mistakes to Avoid. In *ATM Forum/94-0882*, 1994.
[18] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols. In *Proc. of WCNC*, Mar 2005.
[19] K. Sundaresan, V. Anantharaman, H. Y. Hsieh, and R. Sivakumar. ATP: A Reliable Transport Protocol for Ad Hoc Networks. In *ACM Mobihoc*, 2003.

APPENDIX

A. WEBPAGE

Detailed instructions for implementing a MiNT node and integrating the testbed is available at: http://www.ecsl.cs.sunysb.edu/mint.